COMSC-210 Lecture Topic 14
n log(n) Sorting Algorithms

Reference
Childs Ch. 14
sorting-algorithms.com

No Holes
sort up to a "fill-line"

The "Heapsort" Algorithm
uses a binary "tree" structure
(stored in array, but thought of as a binary tree)
first: arrange array into a heap (highest value at top)
then: repeatedly remove top of heap
  to its position in the array
  starting from the end,
  until the heap is empty.
best suited to arrays

Viewing An Array As A "Heap"
heaps are *arrays*, but all arrays are not heaps
"heap" is another way of looking at an array, as is "table"
each position in array corresponds to a
  position in a binary tree

a[0] is the top; a[1] and a[2] are its children
a[3] and a[4] are children of a[1]
  a[5] and a[6] are children of a[2]
...and so it continues, doubling the #of children
  in each successive generation
the children of a[i] are a[2i+1] and a[2i+2]

Building A Heap
in any array, all elements from n/2 to n-1
  are valid heaps: no children
but including n/2-1, it may no longer be a heap
  because it has children
so working from n/2-1 to 0, one-at-a-time,
  consider the array from there to n-1 as a heap,
  remove the top, promote remaining elements,
    and reinsert the removed top
after removing a[0] and reinserting, we have heap!
click here and here for array code sample

Sorting The Heap
one-by-one, fill from the back of the array
top of heap is highest value: belongs at end of array
so: (1) save value at end of array,
  (2) copy top to end of array
  (3) reinsert saved value into heap
repeat until < 3 are left in heap, array is sorted
each cycle, heap size in left half of array is *decreased by 1*
  and the sorted array size in the right half is *increased by 1*
click here for array code sample
click here for tree-based heapsort visualization

The "Quicksort" Algorithm
arrayed or linked
public function: `void quickSort(int);` // #of elements to sort
divide values into a pivot and 2 separate collections
swap values so that:
  ...all values in 1st collection are *less than* the pivot
  ...all values in 2nd collection are *greater*

concatenate 1st collection, pivot, and 2nd collection
recurse 1st and 2nd collections until
  ...each has *size=1*
suitable for arrays *and* linked-lists
click here for list code sample
click here for array code sample

The "Mergesort" Algorithm
divide values into separate collections
subdivide each collection
repeat subdividing until each collection has *size=1*
merge collections into one big collection
suitable for arrays and linked-lists
for lists with *more than one* value:
click here for code sample

O(n) Sorting!
radix (or "bogo") sort (ref)
click here and here for radix sort visualizations